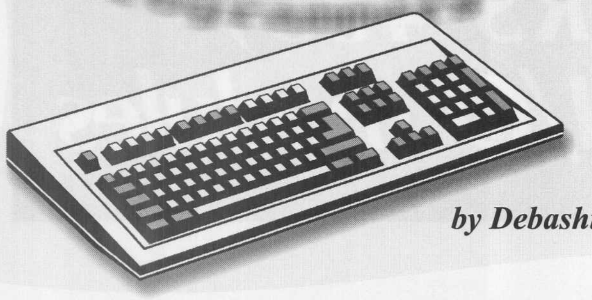
Programmers' Corner



by Debashis Ghosh

31	23	15	7	
Device ID		Vendor ID		00
Base address register #0				10
Base address register #1				14
Base address register #2				18
Base address register #3				1C
Base address register #4				20
			Interrupt line	3C

Introduction

The PCI bus offers simple "plug and play" capability to the end user in addition to other major advantages over the ISA bus. Users benefit because there are no switches or jumpers to be set in selecting the base address and interrupt required by a PCI data acquisition board like the PCI-DAS1602/16. The PCI BIOS makes these assignments when the system boots, thereby guaranteeing that the your software can access the board. Contrast this with the ISA bus, where either a misplaced base address switch or a base address that conflicted with other hardware meant that one could not access the board in question.

How to determine the base address and interrupt of a PCI board

Since the base addresses of various registers in a board are set by the PCI BIOS, you will need to query the BIOS to determine these. (This is done automatically by our Universal Library as well as all upper-level application software.) This must be done before you access and/or program the board in any way. Before delving into the details we need to look at the PCI Configuration Header Space.

Each PCI bus device contains a unique 256-byte region called its configuration header space. The organization of this space is given in the table below. The important fields that are relevant to this discussion are named; the others are either reserved or not used.

The device and vendor IDs are unique values that are assigned to each PCI data acquisition board. These are used in querying the PCI BIOS for a specific board. The base address registers provide a mechanism for assigning I/O space. Registers in ComputerBoards' family of PCI boards are offset from the I/O address contained in a specific base address register. For example, in the PCI-DAS1602/16 the ADC data can be read at the I/O address contained in base address register 2 and the DAC can be updated by writing to the I/O address contained in base address register 4.

The following code snippets describe the steps necessary to determine the contents of the various base address registers. The assembly language code is for Windows 3.1 or DOS. For the sake of clarity and brevity error handling code is not shown.

```
; ;PCI function values
PCI_FUNC_ID         equ 0b1h
FIND_PCI_DEV        equ 02h
READ_CONFIG_BYTE    equ 08h
READ_CONFIG_WORD    equ 09h
READ_CONFIG_DWORD   equ 0ah

;Configuration register offsets
VID_ADDR    equ 00h
DID_ADDR    equ 02h
BADR0_ADDR  equ 10h
BADR1_ADDR  equ 14h
BADR2_ADDR  equ 18h
BADR3_ADDR  equ 1ch
BADR4_ADDR  equ 20h
INTLN_ADDR  equ 3ch

; Operation registers offsets
INTCSR_ADDR equ 38h
BMCSR_ADDR  equ 3ch

;Misc. equates
IO_ADDR_MASK    equ 0fffch
INTCSR_DWORD equ 00FF1F00H ; dword written to INTCSR_ADDR
BMCSR_DWORD equ 08000000H ; dword written to BMCSR_ADDR

;***************************************************************
; Macro to write a 32-bit dword to an I/O port. This is called
; from 16-bit code segment.
; Arguments : dwValue = DWORD value to write to port
; Uses      : dx must have the port address
; Returns   ; eax=dwValue
; Destroys  : eax
;***************************************************************
OUTD MACRO dwValue
 db 66h   ; OPSIZE:
 db 0b8h; mov eax
 dd dwValue          ; OPSIZE : mov eax, dwValue
 db 66h, 0efh        ; OPSIZE : out dx, eax
ENDM
```

```
;STEP 1 : Find the specific board.
mov   dx, VENDOR_ID         ; vendor id available from ComputerBoards
mov   cx, PCI_DAS1602_DID          ; device id available from ComputerBoards
mov   si, 0
mov   ah, PCI_FUNC_ID       ;load pci function id into AH
mov   al, FIND_PCI_DEV      ;load search for device function into AL
int   1ah          ;call bios  ; Output:   bl = device & function number
      ;                        bh = bus number
      ;                        CF set if error

;STEP 2 : Read the base address registers, using BX obtained earlier
mov   di, BADR0_ADDR        ;read base addr0
mov   ah, PCI_FUNC_ID       ;load PCI function id
mov   al, READ_CONFIG_DWORD    ;load read double word function
int   1ah          ;call bios ; Output:   ecx= register value, CF set on error
and   cx, IO_ADDR_MASK
;At this point CX has the address contained in base address register 0. Repeat the above five instructions with the offsets of ;the other base address register in DI. You can store the contents of CX in some variable to avoid having to access the ;BIOS repeatedly.

; STEP 3 Find the interrupt used by the board.
mov   di,INTLN_ADDR         ;read the interrupt line reg
mov   ah, PCI_FUNC_ID       ;load PCI function id
mov   al, READ_CONFIG_BYTE        ;load read byte function
int   1ah          ;bios interrupt call, returns ch zero, cl is IRQ#

;STEP 4 : Reset and enable interrupts if the board uses interrupts.
; First, copy base address 0 obtained earlier into DX
add    dx, BMCSR_ADDR
OUTD   BMCSR_DWORD          ;reset mail box interrupts
sub    dx, 4                ;dx=INTCSR_ADDR
OUTD   INTCSR_DWORD         ;reset pending interrupts and enable interrupts
```

The OUTD macro allows 32-bit I/O write to the Interrupt Control/Status register and Bus Master Control/Status Register.

(RTD continued from page 1)

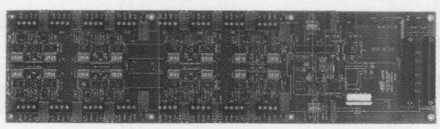
many standards that continue to exist on parallel planes, the industry has largely now standardized on the European/DIN model with alpha = 0.00385. Multiplying the RTD alpha by its standard resistance at 0 °C provides a scale factor in ohms/°C. The typical unit is produced with a resistance of precisely 100 ohms at 0 °C, though there are Pt RTDs available with different base resistances. In the case of the standard 100 ohm, 0.00385, platinum RTD, the actual output scale factor is 100 x 0.00385 or 0.385 ohm/°C

The transfer function of an RTD is not perfectly linear. In fact, the 0.00385 RTD scale factor varies from 0.44 ohm/°C at very low temperatures (-200 °C) to 0.29 ohm/°C at high temperatures (850 °C). There is a standard polynomial conversion formula that is easily implemented in software. However, for the typical system operating in the 0- to 100-°C range, simply using a straight 0.385 ohm/°C will provide better than ±1 °C accuracy, and is often sufficient.

The RTD is the second most commonly used temperature measurement sensor. Only the thermocouple rates as more popular. When compared to the thermocouple, the RTD provides better inherent accuracy, has a more linear transfer function, provides almost as wide a temperature measurement range, and avoids the problems of the cold-junction thermocouple. Thermocouple *system* error specifications typically fall in the range of ±3 °C or ±5.4 °F. Worst case RTD errors in the -50 to +50 °C range can typically be kept to under ±1°C, with repeatability of better than 0.1 °C.

The RTD also offers a larger output than the thermocouple. While the output of a J-type thermocouple is in the range of 50 µV/°C, a 100-ohm Pt RTD provided with a 1-mA excitation current has a scale factor almost ten times greater at 385 µV/°C. In a perfect world, this would make very little difference because you can simply increase the gain of your input system and obtain the same temperature resolution in either case. However, in the real, noisy world, the ability to run your RTD input gain at a factor of ten less than for a thermocouple makes the RTD system much more immune to ambient electronic noise.

(continued on page 11)



ComputerBoards' CIO-EXP-RTD16 provides the signal conditioning required to connect up to 16 RTDs to any of our standard data acquistion boards.